Mobile Task Offloading Under Unreliable Edge Performance

Md Rajib Hossen
University of Texas at Arlington

Mohammad A. Islam
University of Texas at Arlington

ABSTRACT

Offloading resource-hungry tasks from mobile devices to an edge server has been explored recently to improve task completion time as well as save battery energy. The low latency computing resource from edge servers are a perfect companion to realize such task offloading. However, edge servers may suffer from unreliable performance due to its rapid workload variation and reliance on intermittent renewable energy. Further, batteries in mobile devices make online optimum offloading decisions challenging since it intertwines offloading decisions across different tasks. In this paper, we propose a deep Q-learning based task offloading solution, DeepTO, for online task offloading. DeepTO learns edge server performance in a model-free manner and takes future battery needs of the mobile device into account. Using a simulation-based evaluation, we show that DeepTO can perform close to the optimum solution that has complete future knowledge.

1. INTRODUCTION

The increasing adoption of user-level artificial intelligence (e.g., smart assistant) is pushing more and more resource-hungry applications to mobile devices. These devices, however, are heavily resource-constrained systems as they are equipped with slower processors and powered by batteries with limited capacity. Consequently, to alleviate the computing burdens, offloading demanding tasks/applications from mobile devices to a computer server has been actively explored [1, 10]. Towards that the emerging edge computing, with its low-latency computing resource at the Internet's edge, is a perfect companion to realize mobile task offloading to edge servers.

Edge servers, however, may suffer from unreliable performance as they are subject to more rapidly varying workload than larger data centers with many servers [8]. This is because edge servers are spread geographically and have small serving areas. Therefore, they can experience a larger workload change due to a few users. Besides, many edge servers/data centers utilize renewable energy sources (e.g., solar panels) to supplement their power demand [6]. The intermittent nature of renewable energy can also limit edge server's capacity and hence its performance.

When the edge server is experiencing a workload spike or the server capacity is capped due to limited renewable generation, its service performance is adversely affected. To tackle this performance degradation, as shown in Fig. 1, edge computing comes with fail-safe options that route workloads to other low utilized edge servers or the cloud [1, 10]. However, existing literature assume that an offloaded task bound to the edge server will always be processed at the edge [12, 7] and focus mainly on capacity constraints to optimize task offloading. More importantly, the mobile device remains oblivious of this edge-to-cloud offloading which may increase task completion time due to the added communication and data transfer time between the edge server and the cloud.

Figure 1: Mobile devices offload resource hungry tasks to the edge server. The edge server further offloads tasks to cloud during high workloads.

Optimizing task offloading under unreliable edge performance along with the limited battery is challenging. First, all task offloading decisions are coupled since they affect the battery energy. For instance, a task processed at the mobile device now uses up battery energy and may force future tasks to offload due to limited available battery. Second, the unreliable edge performance makes it difficult to accurately estimate the task completion time when a task is offloaded to the edge. Further, the edge-to-cloud offloading varies with different edge servers due to their diverse operating environments. Hence, rather than using a generalized approach, it needs to be individually estimated at each edge server.

To tackle both these challenges, we propose a deep Q-learning based solution called DeepTO. It decides task offloading in an online fashion and incorporates future tasks into account by using a Markov decision process (MDP) formulation. Moreover, DeepTO learns the edge-to-cloud offloading behavior in a model-free manner based on its environmental feedback. Our choice of a learning-based solution is motivated under the assumption that, *even though invisible to a mobile device, the edge-to-cloud offloading is done following some policy at the edge server, and therefore, this policy can be learned and utilized in making better offloading decisions.* We conduct a simulation-based evaluation with a mix of real-world and synthetic traces, and we find that DeepTO performs close to the optimum solution with complete future information.

2. MOBILE TASK OFFLOADING

Here we formalize our offloading problem for the mobile device and then present our solution. We do not incorporate the edge server management (and hence the edge-to-cloud offloading) which is not a part of mobile's offloading decision.

Workshop on AI in Networks and Distributed Systems (WAIN) 2020 Milan,Italy

2.1 Problem Formulation

We consider the k-th offloading task in the mobile device is represented using a tuple $a_k = \{c_k, d_k\}$ where c_k is the task's computation requirement and d_k is its data size. For each offloading task, the mobile device can either process it on the mobile device or offload it to an edge server.[1] We use an event-triggered model where every new offloading task invokes an offloading decision. The offloading decision for task a_k is represented by $x_k \in \{0, 1\}$ where $x_k = 0$ indicates a_k will be processed on the local device and $x_k = 1$ indicates a_k will be offloaded to the edge server.

Task completion time. In our context, the task completion time consists of both the processing time and data transmission time. The processing time of task a_k depends on its processing requirement (c_k) and the allocated computing resource and can be represented as

$$t_k^{pr} = (1 - x_k) \cdot \frac{c_k}{F^m(\mu_k^m)} + x_k \cdot \frac{c_k}{F^e(\mu_k^e)} \tag{1}$$

where μ_k^m is the mobile device utilization and μ_k^e is edge server utilization, while $F^m(\mu_k^m)$ and $F^e(\mu_k^e)$, functions of utilization, are the allocated computing resources in cycles per unit time on the local device and the edge server, respectively. The data transmission time, on the other hand, depends on the size of data d_k and available transmission rate of the uplink and can be represented as

$$t_k^{tx} = x_k \cdot \frac{d_k}{u_k} \tag{2}$$

where u_k is the uplink rate at the time when task a_k is being processed. u_k varies with time and depends on the transmission medium (e.g., WiFi vs 4G/LTE), condition of the wireless channel, and network traffic. We do not add the time required for downloading the results from the edge server since the result returned are typically not data-intensive [2]. Combining the processing time and transmission time together we write task completion time for task a_k as

$$t_k = t_k^{pr} + t_k^{tx} \tag{3}$$

Energy consumption. As in our task completion time, the energy consumption also consists of two parts - processing energy and data transmission energy. Here we are only concerned about the energy consumption of the mobile device and hence edge offloaded tasks consume no processing energy whereas tasks processed on the mobile device do not incur any transmission energy consumption. The energy consumption of task a_k can be expressed as

$$e_k = (1 - x_k) \cdot e_k^{pr}(f_k^m) \cdot t_k^{pr} + x_k \cdot P_k^{tx}(u_k) \cdot t_k^{tx} \tag{4}$$

where, $e_k^{pr}(f_k^m)$ is the energy consumption rate for occupying computing resource $F^m(\mu_k^m)$ on the mobile device and $P_k^{tr}(u_k)$ is the transmission power of the mobile device for a transmission rate of u_k.

Objective. Our objective is to optimize the task offloading decision at the mobile device. This optimization is a multi-objective problem where we want to minimize the task completion time and energy consumption. We formalize this as the following optimization problem OPTO (**OP**timum **T**ask **O**ffloading).

$$\text{OPTO}: \quad \underset{x_k}{\text{minimize}} \sum_i (t_k + w \cdot e_k) \tag{5}$$

Here, w is the weight parameters for energy consumption and determine how much emphasis OPTO puts on energy consumption over minimizing task completion time.

Challenges. Finding the optimum solution for OPTO is challenging. First, due to the limited battery energy of mobile device, the offloading decision across multiple tasks become intertwined since using energy for processing or data transmission for a current task leaves less energy for future tasks. Therefore, OPTO cannot be solved optimally without prior knowledge of all offloadable tasks' arrival time and resource requirements. Second, due to the edge server's unpredictable performance, during making the offloading decision for task a_k, its task completion time t_k cannot be modeled accurately. Meanwhile, the data uplink rate r_k, and consequently the energy consumption for data transmission also varies with time due to varying channel conditions. Hence, solving OPTO must also incorporate estimation of t_k and e_k. However, these estimation models may vary significantly across different mobile devices as well as edge server and its service locations, making individually modeling for numerous different cases exhaustive. OPTO, therefore, is an ideal problem for a learning-based solution where no such models are required.

Next, we formulate the task offloading problem using a MDP and propose a deep Q-learning based solution which makes offloading decision online and does not require estimation of t_k and e_k (i.e., model-free).

2.2 MDP Formulation

We model our problem as an event-triggered discrete-time MDP where each offloading task starts a new time slot indexed by k. Our system state s_k consists of the task a_k, available battery energy e_k^b, mobile device utilization μ_k^m, edge server utilization μ_k^e, and uplink rate u_k. Note that, while each task a_k's computation requirement c_k and size of data d_k can be known in advance (e.g., image size for a face recognition task), we can seamlessly replace a_k with application id with unknown task sizes. We summarize our MDP as follows:

- System state: $s_k = (a_k, e_k^b, \mu_k^m, \mu_k^e, u_k) \in \mathcal{S}$
- Action: $x_k \in \mathcal{A}(s_k)$
- State transition probability: $P(s_k, x_k, s_{k+1})$
- Reward function: $r_k = R(s_k, x_k, s_{k+1})$

Here, $\mathcal{A}$ is the action space which depends on the current state s_k, $P(s_k, x_k, s_{k+1})$ is the probability of transition from state s_k to s_{k+1} for action x_k, and $R(s_k, x_k, s_{k+1})$ is the reward for taking action x_k at state s_k and moving to s_{k+1}.

Our discrete action space $\mathcal{A}$ restricts an offloading decision from draining mobile battery below zero. On the other hand, while our battery energy e_k^b evolves with the action taken, and therefore follows Markovian assumption, the offloading task a_k and μ_k^m depends on mobile user behavior. Also, instead of the current state and action, change in edge server utilization μ_k^e and uplink rate u_k are determined by external factors. We circumvent this non-Markovian behavior by estimating such state variables for the next time slot

[1] Further offloading from edge to cloud is done by the edge and invisible to the mobile.

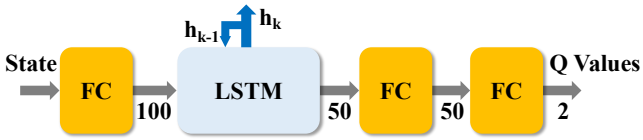


Figure 2: DeepTO's neural network and LSTM layers.

$k+1$ by using a long short-term memory (LSTM) network in our design [3]. Finally, to comply with the optimization objective of OPTO, we set the state transition reward as

$$r_k = R(s_k, x_k, s_{k+1}) = -\left(t_k^* + w \cdot e_k^*\right) \tag{6}$$

Note here that, unlike OPTO, which uses Eq. (3) and (4) to estimate task completion time t_k and energy consumption e_k, respectively, the reward r_k in our MDP formulation is accurately calculated at the end of sate s_k based on actual observed completion time t_k^* and energy consumption e_k^*.

2.3 DeepTO

Here we propose a reinforcement learning-based solution, DeepTO, to solve our MDP problem. In DeepTO, we apply Q-learning which is a widely used reinforcement learning-based solution for solving problems with an unknown environment. In reinforcement learning, an agent learns the best possible actions in every state from the environmental feedback received for its actions.

We use Q-learning as the foundation of DeepTO where the Q-value for a state-action pair represents the "merit" of choosing the action in that state. Then the optimum action policy π^Q can be presented as

$$\pi^Q(s_k) = \underset{x_k \in \mathcal{A}(s_k)}{\text{argmax}}\ Q(s_k, x_k) \tag{7}$$

The main challenge in Q-learning is the estimation of the Q values for every possible state-action pair. Since we have a large state-action space because of the continuous state variables, we use a feed-forward neural network to estimate the Q values associated with the two possible actions, local processing and offload to edge sever, for any state. The neural network takes the state variables as input. We add an LSTM network with a state history vector h_k that estimates future states by utilizing state history saved in h_{k-1}. Fig. 2 shows our implementation and the number of neurons in each fully-connected layer.

The deep neural network, also called Deep-Q-Network (DQN), estimates the Q values as $Q(s, x|\theta)$ where θ is the weight parameters of the neural network and can be learned using gradient descent optimizer. The loss function of DQN for a data set $\mathcal{D}$ is expressed as

$$L(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s_k, x_k, s_{k+1}) \in \mathcal{D}} \left(y_k - Q(s_k, x_k|\theta)\right)^2 \tag{8}$$

where y_k is the network target calculated using a discount factor γ as

$$y_k = r_k + \gamma \max_{x_k \in \mathcal{A}(s_k)} Q(s_{k+1}, x_k|\theta) \tag{9}$$

For faster convergence during training, we also use experience replay buffer, mini-batch gradient descent, and ϵ-state-action exploration [3]. We omit discussing these DQN enhancements due to space limitation.

3. EVALUATION

In this section, we first discuss our evaluation settings, performance metrics, and benchmark algorithms. We then present our results.

3.1 Methodology

Settings. We use a simulation-based evaluation of DeepTO using a mix of real-world and synthetic traces. We consider a mobile device with a battery capacity of 120 Jules, 1.5 GHz processor, and a transmission power of 500 mW [5]. The arriving offloading tasks have uniformly randomly chosen data sizes between 500 KB and 1 MB, and processing requirements between 500 million and 2000 million CPU cycles. To incorporate other tasks/applications running on the mobile device, we set the available capacity for the offload table tasks between 20% to 100% of the 1.5GHz (i.e., 1500 million CPU cycles per second) processor. We calculate the energy consumption for task processing as $\tau * F^m(\mu_k^m)^2 * c_k$ where τ is the effective switched capacitance of the chip. We use $\tau = 10^{-27}$ in our simulation [4]. To account for the energy consumption of other non-offloading tasks, we consider a constant energy consumption rate that uses up the battery in 48 hours.

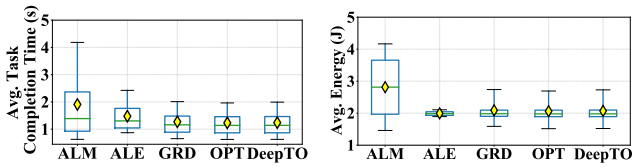
For both the edge and cloud, we consider a 3.6 GHz processor. We use Uber's regional rideshare requests in New York as a typical workload pattern of an edge server [8]. The available processing capacity at the edge with utilization $\mu^e \in [0, 1]$ is calculated as $(1 - \mu^e) * 3.6 \times 10^9$ cycles/seconds. To account for the unreliable edge performance, we offload tasks from edge to cloud with a probability $2.5 \cdot max(\mu^e - 0.5, 0)$. We consider the cloud always has its full capacity (i.e., 3.6×10^9 cycles/seconds) available for the offloaded task, but incurs additional network delay of ∼200 milliseconds based on our ping latency data collected from Azure US-South-Central callback server. We choose the varying uplink rate between 8 Mbps and 13 Mbps. While synthetic, we choose our setting based on other recent relevant studies [11, 9, 4, 5].

We implement DQN using Keras and utilize our edge server trace and mobile tasks to train DeepTO for 5000 episodes. We use a weight parameter $w = 0.7$ to have approximately equal emphasis for task completion time and energy consumption. We use learning rate $\alpha = 0.001$, and discount factor $\gamma = 0.9$. We dynamically change the state-action exploration variable ϵ that starts from 1 with a decrement of 0.0005 per episode and settles at 0.05.

Performance metrics and benchmarks. We use average task completion time and average energy consumption of the offloading tasks as the performance metrics. We compare DeepTO with four benchmark policies. The ALM policy process every task on the mobile device. ALE, on the other hand, always offload tasks to the edge. GRD is oblivious of edge server's further task offloading to the cloud and makes offload decisions to minimize (5) independently for each task. It also does not consider the battery energy level during the offloading decision. However, we restrict all local processing for GRD when the remaining battery is less than 20%. OPT is the optimum policy with complete future knowledge that minimizes OPTO and acts as the performance upper bound for our evaluation.

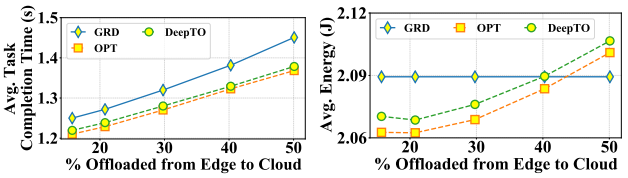
3.2 Results

We show the average task completion time of the offloaded



(a) (b)

Figure 3: Performance of offloading tasks under different offloading strategies. (a) Average tasks completion time. (b) Average energy consumption.



(a) (b)

Figure 4: Impact of increasing unreliability of edge-performance (i.e., more edge-to-cloud offloading) on task offloading.

tasks for different policies in Fig. 3(a) where the boxes represent the values between the lower and upper quartile (i.e., 25% to 75%), the line inside the box is the median, the diamond markers are the mean, and the whiskers extend up to the lower 10-percentile and the upper 90-percentile values. We see that ALM has the worst average performance due to its limited processing capacity. ALE does better than ALM since it utilizes powerful edge servers for task processing. But, it incurs the data transmission time. GRD performs better than ALM and ALE as it can choose between local processing and edge offloading based on a task's computing requirement and data size. However, it performs worse than OPT and DeepTO because it is oblivious to edge-to-cloud offloading resulting in inaccurate estimation of task completion time. DeepTO, on the other hand, performs close to OPT as it learns the edge's cloud offloading behavior and can avoid cases where edge-to-cloud offloading affects performance. In terms of energy consumption shown in Fig. 3(b), GRD is not much worse than OPT or DeepTO since the edge-to-cloud offload mainly affects task completion time and does not add additional energy consumption for the mobile device. ALE has the best energy performance since it only incurs data transmission energy, while ALM has the worst energy consumption performance.

Next, in Fig. 4 we look at how the performance of the offloading strategies change as the edge server's performance unreliability increases. For that, we increase our edge-to-cloud offload probability and compare DeepTO with OPT and GRD for different percentages of tasks offloaded from the edge server to the cloud. We see in Fig. 4(a) that DeepTO can maintain performance close to the OPT even when the edge-to-cloud offloading increases. GRD, on the hand, drifts farther away from OPT since it makes more inaccurate predictions with increasing edge-to-cloud offloading. The energy consumption in Fig. 4(b), on the other hand, shows that GRD's energy consumption does not change with more edge-to-cloud offloading since it only affects the task completion time. Energy consumption of both OPT and DeepTO changes with offloading percentage and when edge-to-cloud offloading goes beyond ∼40%, both consume more energy than GRD as they process more task on the mobile device.

We also evaluate DeepTO with different simulation parameters and settings. The results we obtain are consistent with that of Figs. 3 and 4.

4. CONCLUDING REMARKS

In this paper, we investigated how a deep reinforcement learning-based solution makes mobile task offloading decisions when the operating environment exhibits unreliable behavior. We showed that our deep Q-learning based solution, DeepTO, can perform very close to the optimum offloading algorithm with complete knowledge of the future task and operation environment. We also showed that DeepTO can evolve with increasing unreliability of edge servers and maintain a close to optimum performance.

5. REFERENCES

[1] Abdulhameed Alelaiwi. An efficient method of computation offloading in an edge cloud platform. *Journal of Parallel and Distributed Computing*, 127:58–64, 2019.

[2] X. Chen, L. Jiao, W. Li, and X. Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, October 2016.

[3] Bingqian Du, Chuan Wu, and Zhiyi Huang. Learning resource allocation and pricing for cloud profit maximization. In *AAAI*, 2019.

[4] Songtao Guo, Bin Xiao, Yuanyuan Yang, and Yang Yang. Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. In *INFOCOM*, 2016.

[5] Ji Li, Hui Gao, Tiejun Lv, and Yueming Lu. Deep reinforcement learning based computation offloading and resource allocation for mec. In *WCNC*, 2018.

[6] Wei Li, Ting Yang, Flavia C Delicato, Paulo F Pires, Zahir Tari, Samee U Khan, and Albert Y Zomaya. On enabling sustainable edge computing with renewable energy resources. *IEEE Communications Magazine*, 56(5):94–101, 2018.

[7] Ali Shakarami, Mostafa Ghobaei-Arani, and Ali Shahidinejad. A survey on the computation offloading approaches in mobile edge computing: a machine learning-based perspective. *Computer Networks*, page 107496, 2020.

[8] Zhihui Shao, Mohammad A. Islam, and Shaolei Ren. DeepPM: Efficient power management in edge data centers using energy storage. In *IEEE CLOUD*, 2020.

[9] Jinfang Sheng, Jie Hu, Xiaoyu Teng, Bin Wang, and Xiaoxia Pan. Computation offloading strategy in mobile edge computing. *Information*, 10(6):191, 2019.

[10] Liang Tong, Yong Li, and Wei Gao. A hierarchical edge cloud architecture for mobile computing. In *INFOCOM*, 2016.

[11] Shuai Yu, Xin Wang, and Rami Langar. Computation offloading for mobile edge computing: A deep learning approach. In *PIMRC*, 2017.

[12] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang. Deep learning empowered task offloading for mobile edge computing in urban informatics. *IEEE Internet of Things Journal*, 6(5):7635–7647, 2019.